VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017 Lukáš Šťastný

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Šťastný**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. RNDr. Petr Šaloun, Ph.D.**

Konzultant bakalářské práce: Horst Fleischer

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 28. dubna 2017 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla. Především bych chtěl poděkovat panu Horstu Fleischerovi, Diplom-Informatiker (FH), který mě při práci podporoval.

Abstrakt

Tato bakalářská práce se věnuje popisu mé práce ve firmě Tieto Czech s.r.o. v rámci bakalářské praxe. Jako první představím firmu Tieto a moje zařazení v týmu. Dále uvedu použité technologie. Následuje hlavní část, kde popíši úkoly, na kterých jsem pracoval. Úkoly jsou rozděleny na dvě části. První větší část se věnuje práci na Weblistu. Část druhá popisuje práci v platformním týmu.

Klíčová slova: Odborná praxe, Bakalářská práce, C#, DOT.NET, Entity Framework, Javascript, TypeScript, DurandalJS, KnockoutJS, HTML, CSS, Tieto

Abstract

This bachelor thesis aim to describe my work at Tieto Czech s.r.o company during my internship. First, I will introduce Tieto Company and my role within the team I was assigned to. Next, I will mention technologies I used. The main section of this work describes tasks I worked on. Tasks are divided into two parts. First part is devoted to my work on the Weblist information system. Second part then to my work within platform team.

Key Words: Professional practise, Bachelor thesis, C#, DOT.NET, Entity Framework, Javascript, TypeScript, DurandalJS, KnockoutJS, HTML, CSS, Tieto

Obsah

Seznam použitých zkratek a symbolů

API	–	Application programming interface
CSS	–	Cascading style sheets
KPI	–	Key Performance Indicator
MVC	–	Model-View-Controller
ORM	–	Object-relational mapping
SPA	–	Single page application
SQL	–	Structured query language
TIPS	–	Tieto Integrated Paper Solution

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce popisuje mou bakalářskou praxi ve firmě Tieto Czech s.r.o. Ve firmě jsem již od druhé poloviny roku 2015, kdy jsem nastoupil na studentskou stáž jako programátor. V rámci této stáže jsem se podílel na údržbě původní verze tiketovacího systému Weblist. Začátkem druhého čtvrtletí roku 2016 začala renovace tohoto systému.

Některé podkapitoly obsahují části, na kterých jsem pracoval před zahájením bakalářské praxe, která započala v září 2016. Důvodem je, že v rámci bakalářské praxe jsem se také věnoval úpravám některých již dříve hotových úkolů a považuji za vhodné uvést celý kontext práce a ne jen dané úpravy.

Práce je rozdělena na dvě hlavní části. První část se zabývá prací na renovaci tiketovacího systému Weblist. Druhá část pak na práci v platformním týmu.

2 O firmě Tieto

Firma Tieto[1], se sídlem ve finském městě Espoo, je mezinárodní organizace působící v téměř 20 zemích světa. Tieto poskytuje komplexní IT servis. Se svými více než 13 000 zaměstnanci patří mezi největší IT firmy v severní Evropě.

2.1 Mé zařazení

Ve firmě pracuji v oddělení TIPS na pozici C# programátor. Vzhledem k tomu, že v době zahájení mé bakalářské praxe, jsem byl ve firmě již více než rok, nemusel jsem se v rámci praxe seznamovat s interními procesy firmy. Ze začátku, v rámci renovace Weblistu, jsem byl součástí 3 členného týmu ale po dvou měsících byli kolegové převedeni na projekty s vyšší prioritou. Můj tutor Horst Fleischer, Diplom-Informatiker (FH), je vedoucí týmu vyvíjející interní framework TIX, který Weblist využívá. Ke konci praxe jsem přestoupil do jeho týmu a Weblist byl předán dalšímu studentovi.

3 Použité technologie

3.1 Interní framework TIX

Jedná se o interní framework vyvinutý pro renovaci většiny projektů v TIPS. Využívá všechny technologie zmíněné níže. Framework se skládá z těchto vrstev:

- uživatelské rozhraní,
- veřejné API servisy,
- workflow a
- databázová vrstva.

Uživatelské rozhraní je řešeno MVC aplikací v kombinaci s Durandal Frameworkem. HTML je renderováno na serveru pomocí HTML helperů a data jsou poté načteny pomocí webových servis a zobrazena uživateli pomocí Durandal frameworku. O komunikaci s databází se stará Entity Framework.

3.2 C#

Jedná se o moderní objektově orientovaný programovací jazyk vyvinutý firmou Microsoft, který je neustále zdokonalován. Microsoft poskytuje podrobnou online dokumentaci na webu MSDN[2].

3.3 Entity Framework

Entity Framework je open source ORM[3] framework. Byl vyvinut pro snadnější vývoj aplikací využívající databázi. Dokáže strojově přeložit dotaz napsaný v jazyce C# do jazyka SQL.

3.4 Javascript

Javascript[4] je jazyk používaný převážně ve webových prohlížečích. Jazyk podporuje objektově orientovaný přístup, imperativní programování i funkcionální programování.

3.5 Durandal Framework

Durandal[5] je SPA framework napsaný v javascriptu. Durandal je multiplatformní. Díky tomuto frameworku jsou webové aplikace svižnější a část výpočtů je přenesena na stroj klienta díky čemu se odlehčí server. Také pomáhá tvořit lépe organizovaný a čitelný kód.

3.6 KnockoutJS

Durandal narozdíl od jiných SPA frameworků nemá své řešení pro mapování dat na jednotlivé HTML elementy. K tomuto účelů se používá KnockoutJS[6].

3.7 Preprocesory LESS a TypeScript

Preprocesory generují výstup, který je vstupem do jiného programu.

- Typescript je preprocesor pro javascript. Javascript nativně nepodporuje datové typy. V rozsáhlejších projektech to může vést k nečekaným problémům a chybám. Tento problém Typescript vyřeší tím, že nabízí datové typy a můžeme odhalit případné chyby už při kompilaci a ne až za běhu programu.
- LESS je preprocesor pro kaskádové styly (CSS). LESS nabízí prostředky pro definování konstant a vlastních proměnných. Také usnadňuje organizaci vnořených tříd a zpřehledňuje CSS kód.

4 Weblist

Weblist je tiketovací informační systém vyvíjen pro TIPS. Systém je využíván zaměstnanci firmy Tieto a jejich zákazníky. Hlavní funkce systému je zjednodušit a zefektivnit údržbu a podporu nasazených produktů. Komunikace se zákazníky probíhá vytvářením tzv. tiketů, které popisují nějaký problém nebo požadavek. Weblist je synchronizován s dalšími interními nástroji pomocí windows servis a databázových jobů.

4.1 Home stránka

Home stránka (obr. 1) je výchozí stránka Weblistu, která zobrazuje aktuální přehled a stav tiketů. Stránka se každou minutu automaticky aktualizuje. Skládá se z pěti částí:

1. hlavička stránky,
2. aktuální situace - zde jsou tzv. KPI[8] ukazatele,
3. uživatelská sekce - novinky z TIPS sociální sítě,
4. nejnovější tikety a
5. poslední změny.

4.2 TicketDetail stránka

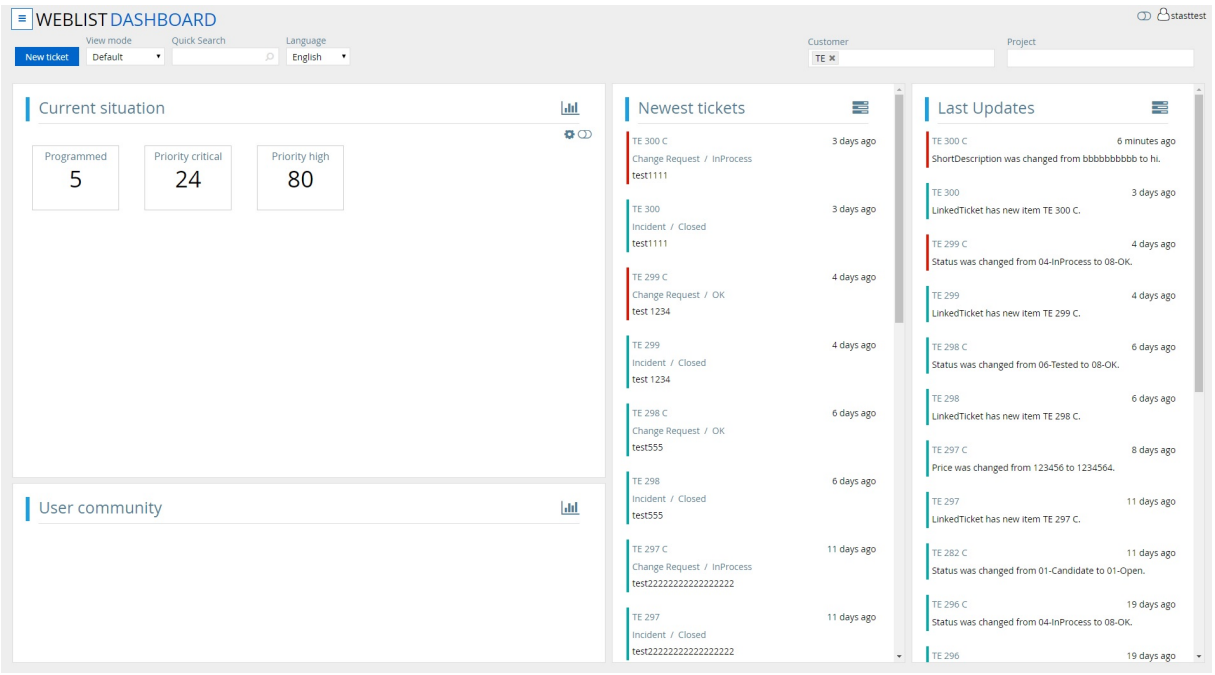
TicketDetail stránka (obr. 2) slouží k zobrazení detailů tiketu a vytváření tiketů nových. Stránka se skládá ze čtyř částí:

1. hlavička stránky,
2. postranní panel - slouží k usnadnění navigace mezi tikety,
3. hlavní část s detaily tiketu a tlačítky pro provádění různých akcí nad tiketem a
4. v poslední části jsou komentáře, historie změn, soubory a přehled vazeb na další tikety.

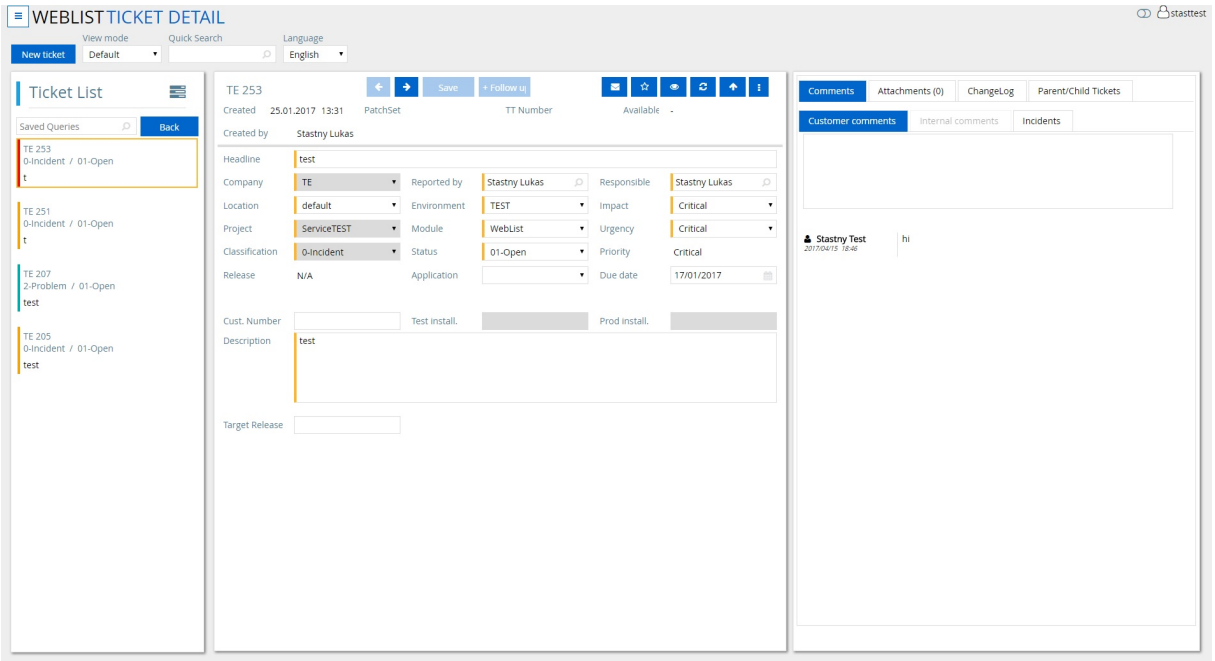
4.3 TicketList stránka

TicketList stránka (obr. 3) slouží k vyhledávání tiketů podle zadaných parametrů. Z této stránky se dá přejít na stránku TicketDetail. Stránka se skládá ze čtyř částí:

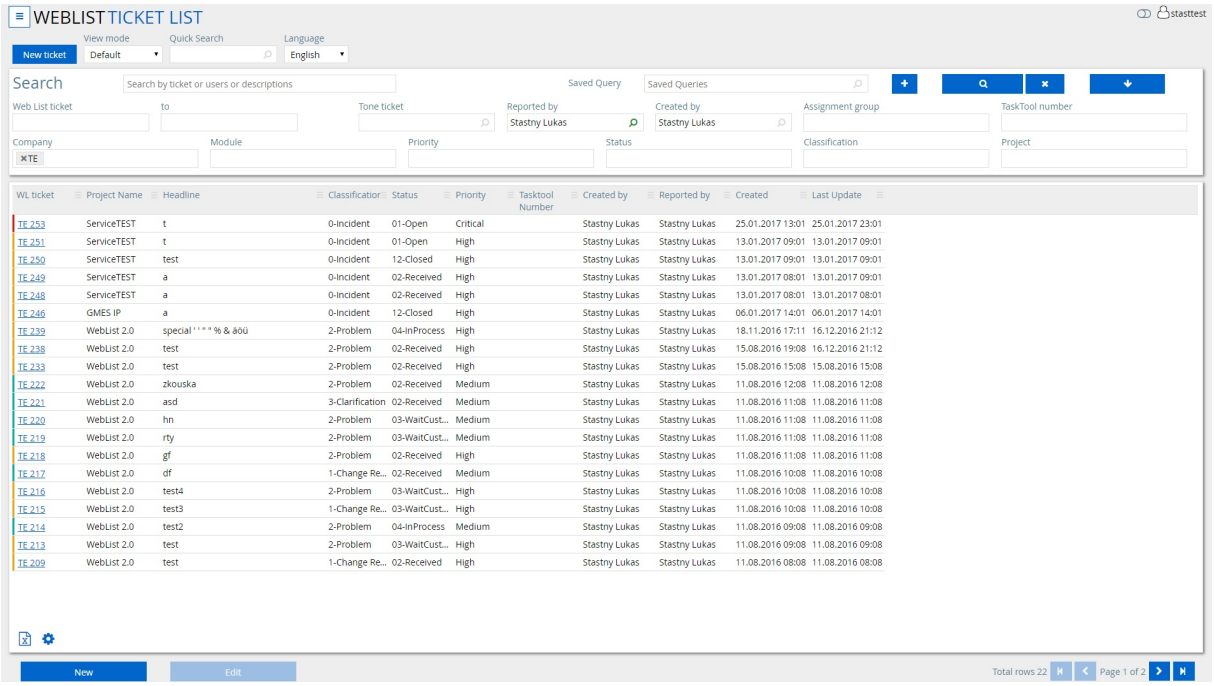
1. hlavička stránky,
2. parametry pro vyhledávání,
3. seznam nalezených tiketů a
4. patička stránky, zde jsou tlačítka pro navigaci.



Obrázek 1: Home stránka



Obrázek 2: TicketDetail stránka



Obrázek 3: TicketList stránka

5 Weblist úkoly

Jednotlivé úkoly jsou zde uvedeny v chronologickém pořadí tak, jak jsem nad nimi pracoval. Jak bylo v úvodu zmíněno, podkapitoly obsahují i části, které jsem implementoval před začátkem bakalářské praxe. Především stránka *TicketList* vznikla již dříve a v částech 5.1.1 až 5.1.4 se věnuji popisu úprav, nad kterými jsem pracoval v rámci praxe. Posílání emailů jsem také vyřešil již dříve a část 5.5 se věnuje převážně změnami implementovanými po zahájení praxe.

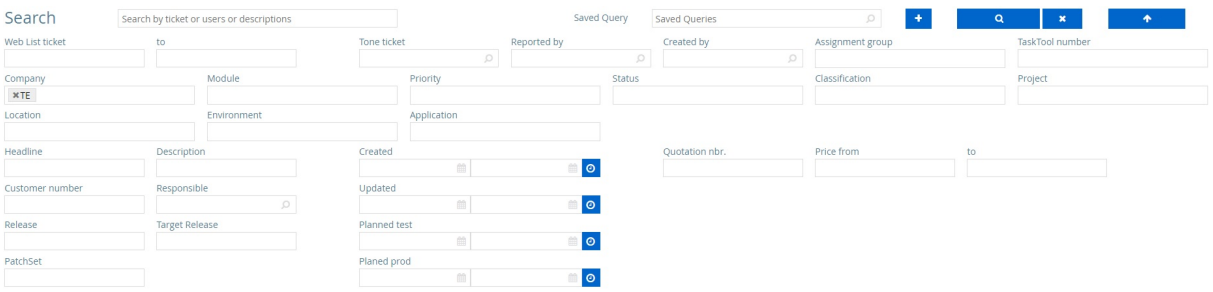
5.1 Implementace stránky TicketList

Jako první úkol jsem měl vytvořit stránku, která bude sloužit pro vyhledávání tiketů (obr. 3). Práci dost zjednodušil fakt, že Weblist nevznikal nově ale šlo o jeho renovaci. Většina vyhledávacích polí je tedy převzata z předešlé verze programu. Na obr. 4 vidíme rozbalený filtr se všemi možnostmi vyhledávání.

Bylo požadavkem, aby vyhledávání fungovalo stejně jako v předešlé verzi Weblistu. K zobrazení výsledků vyhledávání jsem tedy použil HTML helper catalog z TIXu. Catalog funguje tak, že stažená data ze serveru nastránkuje, podle předem definovaného počtu záznamů na stránku. Limitem je, že je nutné definovat maximální počet záznamů. Není praktické tahat ze serveru několik tisíc záznamů najednou. To představovalo problém, jelikož nastavením limitu jsme viděli jen omezený počet výsledků a ne skutečný výsledek z databáze. Proto bylo zapotřebí vytvořit stránkování na serveru.

5.1.1 Serverové stránkování

Výchozí ovládání catalogu jsem schoval a vytvořil nové ovládání, které lze vidět v pravém dolním rohu na obr. 3. Vytvořil jsem novou třídu, která s tímto ovládáním pracuje. Třída obsahuje kolekci s výsledky vyhledávání a proměnné pro samotné ovládání (výpis 1).



Obrázek 4: TicketList stránka - rozbalené vyhledávání

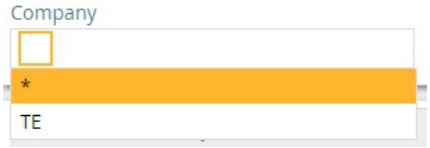
```
public PagedList(IQueryable<T> source, int pageNumber, int pageSize)
{
   this.TotalCount = source.Count();
   this.PageCount = this.GetPageCount(pageSize, this.TotalCount);
   this.PageNumber = pageNumber;
   this.PageSize = pageSize;

   this.Items = new List<T>();
   this.Items
      .AddRange(source.Skip((this.PageNumber <= 1 ? 0 : this.PageNumber - 1) *
         this.PageSize)
      .Take(this.PageSize).ToList());
}
```

Výpis 1: Konstruktor třídy pro serverové stránkování

5.1.2 Možnost vybrat vše

Dále bylo potřeba upravit tzv. chosen helper. Tento helper nám umožnuje vybrat více prvků z předem definovaného seznamu a ve filtru je použit pro výběr společností. V případě Weblistu někteří uživatelé mají na výběr přes 100 společností. Pokud tedy uživatel chce vybrat všechny společnosti kromě např. 15, je vhodné mu nabídnout řešení, které nevyžaduje ručně zaklikávat přes 80 prvků seznamu.

Tento problém jsem vyřešil na klientské straně tím, že jsem dynamicky po načtení dat přidal možnost *vybrat vše* reprezentováno symbolem * (obr. 5).



Obrázek 5: TicketList stránka - volba vybrat vše

Pokud máme vybranou volbu *vybrat vše* a vybereme další prvky seznamu, tyto prvky budou z volby odstraněny. Aby bylo jasné, že vybrané prvky nejsou součástí filtru, jsou vybarveny červeně (obr. 6).

Obrázek 6: TicketList stránka - výběr všech prvků kromě společnosti TE

5.1.3 Grafické znázornění priority tiketu

Každý tiket má svou prioritu. Priority jsou čtyři (Critical, High, Medium, Low). Priorita tiketu určuje jeho naléhavost k vyřešení a je tedy důležitý údaj v zajišťování podpory zákazníků. Pro přehledné rozlišení jednotlivých priorit, jsem je barevně rozlišil a tuto informaci zobrazuji v levém okraji každého řádku výsledku hledání (obr. 3).

Tato funkce není implicitně podporována catalogem a bylo potřeba ji dopsat (výpis 2). Funkce projde všechny řádky výsledku hledání a nastaví jim stín v barvě přiřazenou pro danou prioritu.

```
private colorRowByPriority(tickets: Array<ticketService.ITicket>): void {
   $("td[aria-describedby='weblist-ticketlist-catalog_ticketId']").each((index,
        elem) => {
      var id = Number($(elem).text());
      var ft = Object(tickets.filter((t) => {
      if (t.ticketId === id)
         return true;
      }));

      var color = "#00aba9";

      switch (ft[0].priorityPriorityName) {
         case "Critical":
            color = "#e51400";
            break;
         case "High":
            color = "#f0a30a";
            break;
         }
      $(elem).next("td").css("box-shadow", "inset 3px 0 0 0 " + color);
   });
}
```

Výpis 2: Funkce pro zobrazení barvy priority

5.1.4 Tisk do excel souboru

Stará verze Weblistu nabízela tisk výsledku hledání do excel souboru. Toto sice bylo podporováno frameworkem TIX ale jako v případě stránkování pouze pro data načtena na klientovi. Pro účely Weblistu bylo nutné mít možnost exportovat výsledek všech dat odpovídající nastavení filtru.

Řešením bylo přetížit funkci, která se volala kliknutím na tlačítko s písmenem *X* (vlevo dole na obr. 3). V této funkci jsem vytvořil formulář, který není na stránce viditelný. Pomocí tohoto formuláře posílám požadavek s hodnotou filtru cshtml stránce, kde se vytvoří dotaz na databázi a výsledek se vytiskne do souboru. Díky tomu, že volám cshtml stránku mohu jako odpověď vrátit vygenerovaný excel soubor a webový prohlížeč už za mě tento soubor zpracuje a nabídne uživateli jeho stáhnutí.

Komplikace byla, že catalog nabízí možnost individuálního nastavení sloupců. Nastavit lze jejich pořadí a viditelnost. Požadavkem bylo, aby toto nastavení bylo ve finálním souboru zachováno.

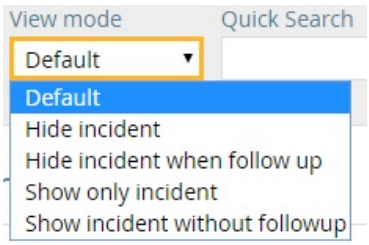
Při volání metody pro vygenerování excel souboru tedy posílám i zobrazené sloupce a jejich pořadí. Poté pomocí reflexe se do souboru vytiskne jejich hodnota (výpis 3).

```
object value = typeof(Ticket)
           .GetProperty(columns[j].PropertyName)
           .GetValue(result[i], null)
       ?? string.Empty;
```

Výpis 3: Reflexe pro získání hodnoty sloupce

5.2 Zobrazovací mód

Tato funkce byla požadována hlavně pro zaměstnance, aby měli větší přehled. Např. některé tikety, které mají vytvořený *follow up* tiket, už není potřeba zobrazovat. Proto bylo navrhnuto implementovat zobrazovací mód (obr. 7). Hodnotu tohoto nastavení berou v potaz všechny funkce napříč celou aplikací. Pro sestavování dotazů filtrů se používá tzv. *Predicate builder*. Volba zobrazovacího módu se přidá do predikátu pomocí statických metod, které s tímto predikátem pracují (výpis 4).



Obrázek 7: Volba zobrazovacího módu

```
switch (criteria.SearchMode)
{
   case TicketSearchMode.HideIncidentWhenFollowUp:
      predicate = predicate.And(Where.Ticket.HideIncidentWhenFollowUp());
        break;
   case TicketSearchMode.HideAllIncidents:
      predicate = predicate.And(Where.Ticket.IsNotIncident());
        break;
   case TicketSearchMode.ShowOnlyIncident:
      predicate = predicate.And(Where.Ticket.IsIncident());
        break;
   case TicketSearchMode.ShowIncidentWithoutFollowUp:
      predicate = predicate.And(Where.Ticket.ShowIncidentWithoutFollowUp());
        break;
   case TicketSearchMode.Default:
      default:
        break;
}
```

Výpis 4: Nastavení zobrazovacího módu

5.3 Postranní panel tiketů

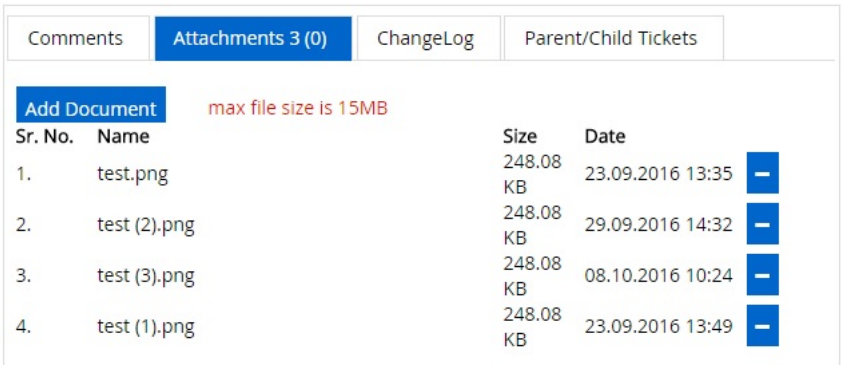
Postranní panel (levá část na obr. 2) slouží k zpřehlednění a zjednodušení navigace mezi tikety. Často se s Weblistem pracuje tak, že se první vyhledají tikety pomocí stránky *TicketList* a následně si zobrazíme detail tiketu pomocí stránky *TicketDetail*. Když se poté chceme přesunout na další tiket nemusíme jít zpět na stránku *TicketList*, protože již máme k dispozici zjednodušený seznam tiketů z předchozího hledání.

Panel také nabízí vyhledávání pomocí uložených filtrů, což je na pozadí stejná funkcionalita jako *SavedQuery* na stránce *TicketList*. K samotnému načtení dat se používá stejná funkce jako na stránce *TicketList*, volána přes servisu, která data zjednoduší.

Při přechodu mezi stránkami by se data ztratila a proto jsou data přenášená pomocí modulu, který je importován do obou *view modelů* stránek. Přenáší se jen hodnoty filtru a při načtení stránky se znovu tikety vyhledají.

5.4 Nahrávání souborů

Nahrávání souborů (obr. 8) bylo v TIXu vyřešeno jen na straně klienta, kdy se pomocí HTML helperu vytvořil formulář. Poté se musela udělat cshtml stránka, která daný formulář zpracovávala. Se soubory se v TIXu moc nepracovalo, proto HTML helper nabízel jen nahrávání a mazání souboru. Stahování jsem musel dodělat a věnuji se tomu v části 6.1.



Obrázek 8: Nahrávání souborů

```
@{
   const string TIPSUI5ViewName = "weblist-ticketdetail";
   string sessionGuid = System.Guid.NewGuid().ToString();
}
```

Výpis 5: Generování jedinečného identifikátoru pro nahrávání souborů

```
private static long lastTimeStamp = RepositoryActions
                        .ConvertToDbTimeZone(TimeProvider.UtcNow).Ticks;
public static long UtcNowTicks
{
   get {
      long original, newValue;
        do {
         original = lastTimeStamp;
            var now = RepositoryActions.ConvertToDbTimeZone(TimeProvider.UtcNow)
                .Ticks;
            newValue = Math.Max(now, original + 1);
      } while (Interlocked.CompareExchange(ref lastTimeStamp, newValue,
          original) != original);
        return newValue;
   }
}
```

Výpis 6: Metoda poskytující čas pro ukládání souborů

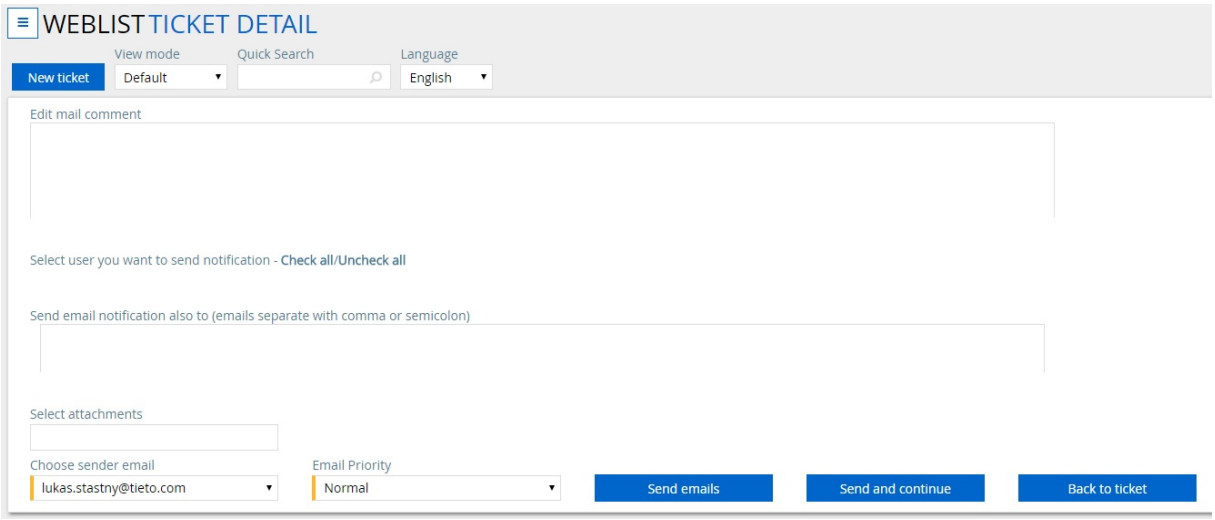
Jedním z důvodu renovace bylo urychlení práce s Weblistem. V případě nahrávání souborů to znamená, že muselo jít nahrát více souborů najednou už při vytváření tiketu. To představovalo problém, protože helper v TIXu při nahrání souboru hned posílal požadavek cshtml stránce, kde se musel zpracovat.

Proto jsem vytvořil tabulku v databázi, do které se dočasně ukládají nahrané soubory. Abych odlišil jednotlivé zdroje, generuji při vytváření stránky *TicketDetail* unikátní identifikátor (výpis 5), který poté vygeneruji do stránky a při posílání souboru na server jej pomocí *jQuery* zahrnu do požadavku. Dočasně uložené soubory se tedy v databázi ukládají pod identifikátorem a k němu přidávám unix čas. Vzhledem k tomu, že TIX helper na nahrávání souborů pracuje paralelně, může dojít ke kolizi a dva soubory by mohly mít stejný identifikátor. Tento problém jsem vyřešil metodou (výpis 6), která poskytuje daný čas a je bezpečná napříč vlákny. Při úspěšném vytvoření tiketu se soubory zkopírují do originální tabulky.

5.5 Posílání notifikací přes email

U některých tiketů se emaily posílají automaticky po vytvoření. U jiných se nabízí stránka (obr. 9), kde lze vybrat komu chceme email poslat. Na obrázku jsou z bezpečnostních důvodu odstraněna všechna jména uživatelů.

Posílání emailů bylo implementováno i ve staré verzi Weblistu, kde uživatelé byli zobrazení vertikálně podle abecedy (obr. 10 vpravo). Uživatelé procházím cyklem v pořadí uvedeném na obr. 10 vlevo a jeden po druhém je generuji do výsledného HTML. K dosažení vertikálního řazení byla ve starém Weblistu použitá funkce obsažena v *ASP.NET WebForms*. Tento luxus jsem neměl k dispozici a tak jsem data musel upravit tak, aby i při klasickém procházení byli uživatelé seřazení vertikálně.



Obrázek 9: Posílání notifikací přes email

A	B	C	A	D	X
D	E	F	B	E	Y
X	Y	Z	C	F	Z

Obrázek 10: Způsob řazení uživatelů

Nejprve data upravuji na serveru, kde zjistím kolik bude řádků a následně je seřadím tak, aby byli abecedně ve sloupcích (výpis 7). Na straně klienta je výhodné mít data uložená ve dvojrozměrném poli logicky rozdělené na řádky a sloupce. Toho jsem docílil vytvořením *computed*[7] proměnné, kterou nabízí framework *KnockoutJS*. Kód můžeme vidět ve výpisu 8. Poté můžu pohodlně pomocí dvojice cyklů vypsat uživatele dle požadovaného řazení.

```
var sortedList = new List<UserCompanyData>();
   int rows = (int)Math.Ceiling((double)userList.Count / columnsCount);
   for (int row = 0; row < rows; row++)
   {
      for (int col = 0; col < columnsCount; col++)
      {
         var current = row + (col * rows) >= userList.Count ? null : userList[
             row + (col * rows)];
         if (current == null)
         {
            sortedList.Add(new UserCompanyData { ShowCheckbox = false });
            continue;
         }
      sortedList.Add(current);
   }
}
```

Výpis 7: Úprava dat pro vertikální řazení (server)

```
private grouped = ko.computed(function () {
   var rows = [], current = [];
   rows.push(current);
   for (var i = 0; i < this.usersToNotify().length; i++) {
      current.push(this.usersToNotify()[i]);
      if (((i + 1) % this.columnsCount) === 0) {
         current = [];
         rows.push(current);
      }
   }
   return rows;
}, this);
```

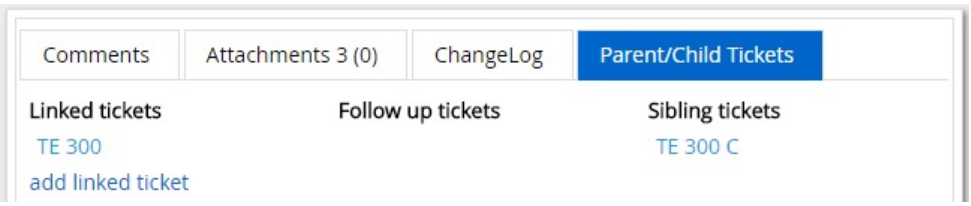
Výpis 8: Úprava dat pro vertikální řazení (klient)

5.6 Zobrazení Parent/Child vazby tiketu

Ve Weblistu většina tiketů může mít tzv. *follow up* tiket. Tento tiket se pak stává tzv. *child* tiket vůči jeho předchůdci. Předchůdce (originální tiket) je *parent* vůči jeho follow up tiketům. Jednotlivé follow up tikety mající stejného předka jsou vůči sobě tzv *sibling* tiketem. Jak vypadá zobrazení těchto vazeb vidíme na obr. 11.

Tato funkcionalita nebyla tak úplně implementována v původní verzi Weblistu. Původně byly zobrazovány jen *Linked tickets*. Jedná se o *parent* tikety ale může se zde připojit i tiket úplně jiný, který vznikl duplicitně nebo tiket s podobným problémem. K usnadnění uživatelům přechod na novou verzi bylo zachováno názvosloví ze staré verze.

Vazby mezi jednotlivými tikety jsou ukládány v samostatné tabulce. Ve výpisu 9 je ukázka sestavení dotazu pro získání *child* tiketů pomocí *Entity Frameworku*.



Obrázek 11: Zobrazení vazby mezi tikety

```
var childs =
   from lt in linkedTickets
   join t in ticket on lt.TicketId equals t.TicketId
   join c in ttCompany on t.CompanyId equals c.CompanyId
   where lt.LinkedToTicket == ticketId
   select new
   {
      TicketId = lt.TicketId,
      WlTicket = t.WebListTicket,
      TicketNumber = t.TicketNumber,
      CompanyName = c.CompanyName,
      LinkedToTicket = lt.LinkedToTicket,
      Id = lt.Id
   };
```

Výpis 9: Zjištění child tiketů

5.7 Implementace zobrazení aktuální situace

Jak už bylo zmíněno, nová verze Weblistu měla zefektivnit práci s tikety. K tomu patří i snížení času reakce na nové tikety. V aktuální situaci jsou zobrazeny KPI[8], přehled posledních změn a seznam nejnovějších tiketů. Informace o aktuální situaci zabírají většinu *Home stránky* (obr. 1).

5.7.1 KPI ukazatele

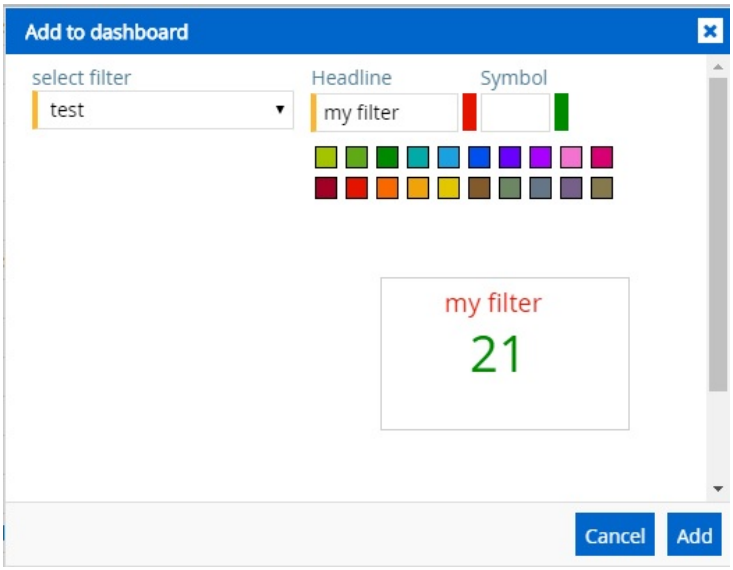
KPI slouží k měření výkonu/stavu nějaké aktivity. V našem případě zjednodušeně měříme kolik tiketů je otevřených, zavřených apod. Tyto hodnoty jsou zobrazeny na obr. 1 vlevo.

Každý uživatel má předem nadefinovaných pár výchozích filtrů, které jsou pro všechny společné. Dále každý uživatel má možnost si nadefinovat svůj vlastní filtr.

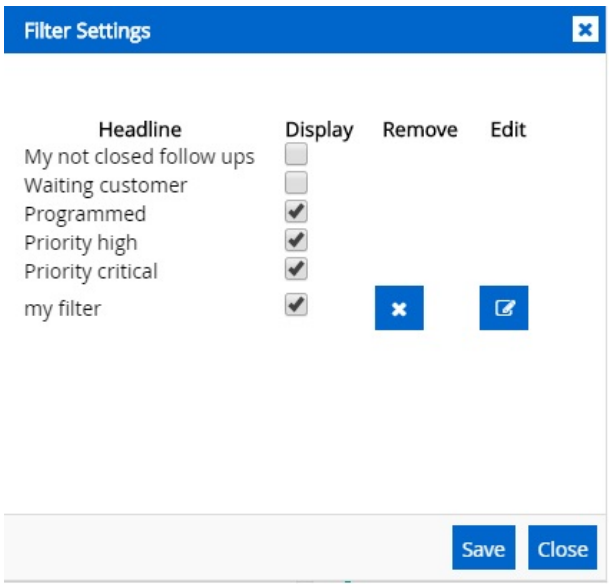
Každý KPI ukazatel reprezentuje výsledek uloženého filtru. Filtrem je na mysli nastavení vyhledávajících parametrů na stránce *TicketList* (obr. 4). Toto nastavení se dá pak uložit pomocí helperu *SavedQuery*. Z takto uloženého filtru lze pak vytvořit KPI pomocí modalního dialogu (obr. 12). U takhle vytvořeného KPI ukazatele můžeme nastavit (obr. 13) jeho viditelnost a taky pomocí tlačítka *Edit* celkové nastavení (obr 12).

KPI využívají dvě TIX komponenty. Jednou je seřaditelný katalog, který uchovává jednotlivé KPI. Druhou komponentou je *SavedQuery*, která načítá uložený filtr z databáze do daných proměnných. *SavedQuery* umožňuje načíst pouze jeden filtr v danou chvíli, proto jsem musel implementovat mechanismus, který načte postupně vše.

Weblist je implementovaný jako SPA a data jsou získávány asynchronně ze serveru pomocí webových servis. Při načtení stránky se tedy servisou zjistí všechny filtry uživatele a poté se jeden po druhém načtou. Načítání probíhá asynchronně, kdy v daný okamžik jsou proměnné uchovávající hodnoty filtru využívány pouze jedním filtrem. Jakmile dojde k zavolání servisy, dané proměnné se uvolní a jsou použity dalším filtrem.



Obrázek 12: Definice nového KPI



Obrázek 13: Nastavení KPI zobrazení

5.7.2 Poslední změny a Nejnovější tikety

Poslední změny (obr. 1 vpravo) zobrazují nejnovější záznamy z tabulky *changelog*. Nejnovější tikety (obr. 1 uprostřed) zobrazují tikety z tabulky *ticket* seřazené podle data vytvoření.

Oba prvky jsou implementované jako tzv. *infinite scroll*. Při dosažení dna se tedy automaticky zavolá funkce, která načte další tikety v pořadí.

```
triggerLoadLatest(event) {
   var elem = event.target;
   if (elem !== undefined) {
      if (elem.offsetHeight + elem.scrollTop >= elem.scrollHeight) {
         this.lastUpdatedProp.pageNumber++;
         this.fetchLastUpdatedTickets.execute(false);
      }
   } else {
      this.lastUpdatedProp.pageNumber++;
      this.fetchLastUpdatedTickets.execute(false);
   }
}
```

Výpis 10: Implementace funkce infinite scroll

5.8 Implementace vytváření tiketů přes API rozhraní

Někteří zákazníci nevyužívají Weblist přímo ale používají své vlastní řešení. Pro tyto zákazníky se muselo vytvořit samostatné API[9] rozhraní. Nebylo možné přímo použít rozhraní, které je používáno grafickým rozhraním, protože implementace na straně zákazníka by byla zbytečně složitá a nepřehledná.

Proto jsem vytvořil rozhraní, které sice používá servisy z grafického rozhraní ale práce s ním je navenek značně zjednodušená. Také motivace byla implementovat rozšířenou validaci a v případě volání s chybnými parametry poskytnout seznam validních možností. Data jsou přenášena ve formátu JSON. O autorizaci se stará samotný TIX framework a není možné tyto servisy bez ověření identity volat. Byly vytvořeny tyto funkce:

- vytvoření tiketu,
- aktualizace tiketu,
- načtení tiketu,
- získaní všech tiketů od určitého časového razítka,

- stažení souboru daného tiketu,
- získaní změn daného tiketu od časového razítka a
- získaní změn množiny tiketů od časového razítka.

Každá metoda obsahuje v odpovědi kolekci *log*, kde každý prvek obsahuje tyto proměnné:

- code - kód chyby,
- name - jméno parametru, kterého se chyba týká,
- text - rozsáhlejší vysvětlení chyby a
- validOptions - seznam validních možností pro daný parametr.

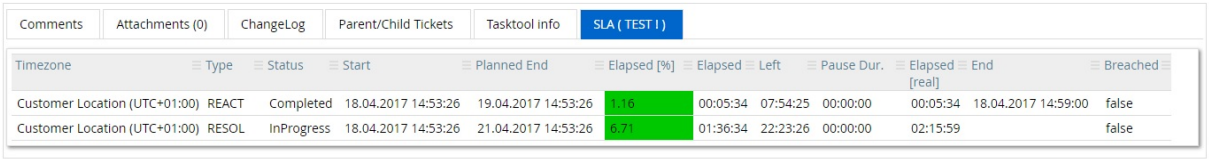
```
"result": {
   "ticket": null,
   "log": [
      {
         "code": -1,
         "name": "Impact",
         "text": "'Impact' with value 'a' is not valid.",
         "validOptions": [
            "Critical",
            "High",
            "Medium",
            "Low"
```

Výpis 11: Ukázka části odpovědi chybného požadavku

5.9 SLA monitorování

V rámci SLA jsou u některých zákazníků definovány intervaly, ve kterých se má reagovat, popřípadě vyřešit daný tiket. Ve staré verzi Weblistu se SLA statistiky počítaly ručně service týmem, což bylo pracné a v rámci renovace bylo požadavkem tento proces zautomatizovat. Jak vypadá implementace vidíme na obr. 14. Monitorovány jsou dva typy SLA:

- reakce a
- vyřešení.



Obrázek 14: Informace o průběhu SLA

Oba typy se vytvoří při vytvoření nového tiketu. Reakce se uzavře jakmile se změní stav tiketu na jiný než *Open*. Vyřešení se uzavře po nastavení stavu, který je v databázi nastaven jako konečný. Nastavením stavu *pause* se běh SLA pozastaví až do opětovné změny stavu.

Výpočty SLA respektují nastavení z databáze, kde jsou definované pracovní hodiny, ve kterých se počítá SLA, a reakční intervaly pro jednotlivé zákazníky. V databázi ukládám pouze časy událostí, které mají vliv na výpočet výsledného času a při načítání dat tiketu se z těchto hodnot dopočítají správné časy vzhledem k aktuálnímu času načtení stránky.

6 Práce v platformním týmu

Ke konci praxe jsem přešel do platformního týmu. Hlavním úkolem týmu je vyvíjet a udržovat TIX framework pro potřeby ostatních týmů. Tento tým je mezinárodní, tvořený programátory z Česka, Německa a Indie.

6.1 Přidání funkcionality pro stahování souborů

Toto byla moje první práce na frameworku. Stačilo jen obohatit původní helper o formulář (výpis 12), který volá cshtml stránku, která vrací daný soubor.

```
_downloadFile: function (url) {
   var formId = "tipsui5-fileupload-download-attachment";
   var form = document.getElementById(formId);
   if (form == null) {
      form = document.createElement("form");
      form.setAttribute("id", formId);
      form.setAttribute("method", "POST");
      form.setAttribute("action", url);
      form.style.visibility = "hidden";
      form.style.height = "0px";
      form.style.width = "0px";
      form.style.borderWidth = "0px";
      document.body.appendChild(form);
   } else {
      form.setAttribute("action", url);
   }
   $(form).submit();
}
```

Výpis 12: Formulář pro stahování souborů

6.2 Zavření menu při kliknutí mimo menu

Tento problém jsem vyřešil z vlastní iniciativy, kdy pro mě bylo frustrující, že se menu zavíralo pouze po kliknutí v navigaci nebo na malé tlačítko pro zavření. V *Durandal Frameworku* je menu implementováno samostatně a je od jednotlivých stránek logicky odděleno. Není moc praktické tedy reagovat na události mimo menu ale musel jsem přijít s řešením implementovatelné přímo v samotném menu.

V menu se nachází pouze navigace a dá se tedy předpokládat, že pokud uživatel menu otevře nemusí být dostupná interakce s původní stránkou. Tohoto faktu jsem využil a při otevření menu překryji původní obsah elementem *div* a nastavím jeho rozměry tak, aby překrývaly celou obrazovku kromě samotného menu. Na tento *div* poté zaregistruji událost reagující na kliknutí myši, ve které menu zavřu a *div* odstraním (výpis 13).

```
sliderWidth:number = $("#tipsui5-pageview-main-menu-slidebar").width();
$sliderClosingDiv = $("<div />").attr("id", "divForSliderClose");

this.$sliderClosingDiv.appendTo("body");
this.$sliderClosingDiv.css({
   "position": "absolute",
   "left": this.sliderWidth,
   "top": "0",
   "width": "100%",
   "height": "100%"
});
this.$sliderClosingDiv.on("click", () => { this.btnCmdClose.execute(); });
```

Výpis 13: Část kódu pro zavření menu

6.3 Autofocus v prohlížeči FireFox

Většina helperů obsahuje parametr pro nastavení vlastnosti *autofocus*. Použití je takové, že tento parametr nastavíme u prvku, který chceme po načtení stránky automaticky označit (většinou např. první vstupní pole). Tato funkcionalita je definována v samotném HTML, problém je ale ten, že implementace této funkce se mezi některými webovými prohlížeči liší.

Vzhledem k tomu, že Weblist je napsaný jako SPA, životní cyklus načítání dat je odlišný od standardního přístupu, kde celá stránka je načtena ze serveru najednou. Toto v prohlížeči FireFox dělalo problém, protože ve FireFoxu se atribut *autofocus* vyhodnocuje pouze při prvotním načtení stránky, kdy v našem případě ještě nemáme k dispozici data. Ty jsou do stránky načtena až po prvotním načtení pomocí webových servis.

Řešením bylo přidat kód (výpis 14), který se vykoná po načtení stránky. Podmínka je obalená ve funkci *setTimeout* z důvodu, že v danou chvíli ještě data stránky nejsou k dispozici. Kdybychom kód vykonali hned, daný prvek s atributem *autofocus* bychom nenašli.

```
if (bowser.name.toLowerCase() === 'firefox') {
   setTimeout(function () {
      $autofocusList = $(context.child).find("[autofocus]");
      if ($autofocusList.length > 0) {
         $autofocusList[0].focus();
      }
   }, 100)
   }
```

Výpis 14: Část kódu řešící autofocus v prohlížeči FireFox

7 Závěr

7.1 Využité znalosti

Během praxe jsem asi nejvíce uplatnil znalosti z předmětů *Programovací jazyky II* a *Architektura technologie .NET*, díky kterým jsem získal větší přehled o pokročilých vlastnostech jazyka C#. Velkým přínosem byl i předmět *Vývoj informačních systémů*, který mi pomohl s lepším pochopením architektury frameworku TIX. Vědomosti získané z předmětů *Úvod do databázových systémů* a *Databázové a informační systémy* mi pomohli s lepší orientaci v databázi.

7.2 Nabyté zkušenosti

Jako největší získanou zkušenost vidím práci v profesionálním prostředí. Vidět jak fungují věci v reálném světě a v mnohem větším měřítku než znám ze školy. Získal jsem také zkušenosti s javascriptem, které jsem využil v předmětu *Vývoj internetových aplikací*. Také zkušenost komunikace v angličtině s týmy z jiných zemí byla velkým přínosem.

7.3 Shrnutí

V rámci praxe jsem se významně podílel na renovaci tiketovacího systému Weblist, který byl úspěšně nasazen do produkčního prostředí. Po skončení praxe zůstávám ve firmě na poloviční úvazek na pozici C# programátor v platformním týmu.

Literatura

[1] Tieto. In: Tieto [online]. 2016 [cit. 2017-04-19]. Dostupné z: https://www.tieto.com

[2] MSDN dokumentace. In: MSDN [online]. United States: Microsoft, 2017 [cit. 2017-04-19]. Dostupné z: https://msdn.microsoft.com/library

[3] MEHTA, Vijay P. Pro LINQ object relational mapping in C# 2008. Berkeley, CA: Apress, c2008. ISBN 9781590599655.

[4] Javascript. In: Javascript [online]. 2016 [cit. 2017-04-19]. Dostupné z: https://www.javascript.com

[5] Durandal dokumentace. In: Durandal [online]. United States: Blue Spire, 2017 [cit. 2017-04-19]. Dostupné z: http://durandaljs.com/docs.html

[6] Knockout dokumentace. In: Knockout [online]. United States: knockout.com, 2017 [cit. 2017-04-19]. Dostupné z: http://knockoutjs.com/documentation/introduction.html

[7] Computed observable. In: Knockout [online]. United States: knockout.com, 2017 [cit. 2017-04-19]. Dostupné z: http://knockoutjs.com/documentation/computedObservables.html

[8] PARMENTER, David. Key performance indicators: developing, implementing, and using winning KPIs. Third edition. ISBN 978-111-9019-831.

[9] NAGEL, Christian. Professional C# 6 and .Net Core 1.0. ISBN 9781119096603.